

# Data Management over Garbled Bloom Filter for Private Set Intersection

Amit Raj Baral

Takeshi Koshiba

Graduate School of Science and Engineering, Saitama University  
s13mm334@mail.saitama-u.ac.jp, koshiba@mail.saitama-u.ac.jp

## Abstract

*Increasing dependency of data availability and the fear of losing the private information during communication motivate the need of privacy protection. One of the common problems occurs when two parties need to privately compute the intersection of their respective sets of data. One or both parties must obtain the intersection (if one exists), while neither should learn anything about the other party set. Till date, one of the efficient approach has been presented with consideration of Big Data by Dong, Chen and Wen (CCS 2013). They have introduced a new data structure named as Garbled Bloom Filter (GBF), an improved Bloom filter, for the computation of the Private Set Intersection (PSI). While the PSI protocol (DCW13 protocol) by Dong, Chen and Wen is pretty efficient, it does not support for managing dynamic data. In this paper, we present data management techniques on the server side of the DCW13 protocol, which compliment the DCW13 protocol.*

## 1. Introduction

In today's world, the privacy of data is a very precious asset. There are many realistic scenarios where the private data must be shared among mutually suspicious entities for the communication. Let us consider a few examples:

1. A government agency wants to confirm that employees of its industrial contractor have not any criminal records. Neither the agency nor the contractor is ready to share their respective data sets but both would like to know if there is any intersection between them.
2. National law enforcement bodies want to compare and know their respective databases of terrorists, however laws from the country prevent them from revealing the data, still they are allowed to share data on the basis of the common interest.
3. Real estate companies would like to see customers who are dealing with other real estate

companies for their profit, but still they cannot share all the personal details, so the intersection of some mutual information is needed between them.

4. Tax authorities always seek for the information if the tax evaders have accounts with any foreign bank and look to get account information. Bank rules forbid the disclosure, so the tax authority cannot obtain information of suspects.

As the technology is growing rapidly everyday, we are confronting with a huge amount of data in every task we perform. Hackers and spammers are also growing to find the way to leak the private information. Protecting the privacy of data is an always serious issue whole over the world because crackers steal the information and leak it. This has been compulsory to ensure the privacy of data in many countries from their law. We can take examples like HIPPA act in the United States, the European Union Data protection directive and many others. Even though various security techniques have been proposed to enable the privacy preservation but because of the huge size of data and huge processing cost, it is being very difficult to get preserve the data privacy without efficient solutions. In cryptography for the data security, there are different types of techniques presented, among them the Private Set Intersection (PSI) is also a solution. In PSI, we assume two parties, a client and a server, and the client wants to compute the intersection of their private input sets in the manner that at the end the client learns the intersection and the server knows nothing. This PSI problem has been researched extensively because set intersection is a foundational primitive and it also has many practical applications. Efficient PSI protocols have been used as a framework for many privacy oriented applications like bot-net detection [30], Denial of service attack identification [1], online games [8], social networking applications [27] and many others.

Amount of processing data size is also increasing rapidly along with the pace of technology, so finding solutions to handle Big Size of Data processing needs to be considered. Among many other ways

of handing one of the efficient solution will be dynamic management of data. Dynamic management of data helps to increase, decrease or modify in the size of data in the processing. We have extended Dong et. al's protocol [11] to present the different data management techniques over the data structure presented in their paper.

Recently the paper entitled "Faster Private Set Intersection based on OT Extension" has been presented by Pinkas et. al [33], where they have presented that DCW13 protocol can also be enhanced along with increase in performance. They have presented the computation complexity can be reduced by half if the Oblivious Transfer mechanism used in the protocol is optimized. Our dynamic data management techniques can even be applied with this optimized approach of PSI protocol because techniques we have presented doesn't deal with the workflow of the protocol rather deals on the server for preparation of GBF.

### 1.1. Our Contribution

In this paper, we are considering the protocol presented from Dong et al. [11] where they have used a new variant of Bloom filter named as Garbled Bloom Filter. After they presented the protocol, it is considered to be more efficient than existing any other PSI protocols, so it motivated us to add further improvement to it with adding the data management methods. We present several data management methods that can be implemented on the Garbled Bloom Filter (GBF). We show properties of the GBF that can be carried out with clear algorithmic explanation as well as many researchers have considered different dynamic data management properties of the standard Bloom filter and the counting Bloom filter. By using the properties on the GBF, we illustrate data addition on the GBF, data removal within the GBF and merging two GBFs into a single GBF.

### 1.2. Organization

This paper is organized as follows. In Section 2, we have explained related works. We borrow notations and definitions from Dong et al. [11] and show them in Section 3, which we also use in this paper. We review the DCW13 protocol in Section 4. We propose our data management methods in Section 5 along with execution algorithms. We conclude the paper in Section 7.

## 2. Related Work

The concept of the Private Set Intersection was introduced by Freedman et al. [18] in 2004. Their

protocol is proven secure against semi-honest adversaries in the standard model and can be extended for malicious adversaries in the Random Oracle model at an increased cost. After them, Kissner and Song proposed PSI protocols in multiparty settings [26]. Hazay and Nissim proposed protocols which are more efficient even in the presence of malicious adversaries. They recently present an improved construction of the oblivious polynomial evaluation based PSI [21]. Dachman-Soled et al. [15] present an improved PSI construction. Their construction incorporates a secret sharing of polynomial inputs. Since Shamir's secret sharing [36] implies Reed Solomon codes, they do not need generic zero-knowledge proofs. Hazey and Lindell also proposed different approach for PSI which is based on oblivious pseudorandom function evaluation [20]. Jarecki and Liu improve the previous solution, proposing a protocol secure in the standard model against both malicious parties, based on the decisional  $q$ -Diffie-Hellman Inversion assumption [23].

Recently, Huang et al. [22] presented a semi-honest PSI protocol based on garbled circuits. Their protocol requires  $O(n \log n)$  symmetric key operations and a small number of public key operations. They have presented that in certain cases this protocol is more efficient than the previous PSI protocols. And under low security settings De Cristofaro's protocol [15] is the fastest, while at high security settings Huang's protocol is more efficient. Recently, Dong et al. [11] proposed a PSI protocol (DCW13 protocol) which is more efficient than these above mentioned existing PSI protocols. They have designed protocol based on a novel two-party communication approach, where a new variant of Bloom filter has been used and termed as Garbled Bloom Filter (GBF) and the approach has been termed as oblivious Bloom intersection. Due to the efficiency, the DCW13 protocol can deal with so called Big Data. Unfortunately, the DCW13 protocol cannot deal with dynamic data. We can consider that Big Data is growing in size. Thus, it is necessary to develop an efficient PSI protocol which can deal with dynamic data. In future, it might be necessary for the applications to handle dynamic data not only because of Big Data size but also to carry out comparison in performance value and to determine top most execution power of the server or application.

## 3. Preliminaries

### 3.1. Bloom Filters

A Bloom filter [6] is a space efficient compact data structure that is used to test whether an element is

a member of a set or not. Specifically speaking, a Bloom filter is an array of  $m$  bits that can represent a set  $S$  of at most  $n$  elements. A Bloom filter comes with a set of  $k$  independent uniform hash functions  $H = \{h_0, \dots, h_{k-1}\}$  that each  $h_i$  maps elements to index numbers over the range  $[0, m - 1]$  uniformly. In the Bloom filter, all bits in the array are set to 0 initially. To insert an element  $x \in S$  into the filter, the element is hashed using the  $k$  hash functions to get  $k$  index numbers and these bits at all these indexes in the bit array are set to 1, i.e., set  $BF_S[h_i(x)] = 1$  for  $0 \leq i \leq k - 1$ . To check if an item  $y$  is in  $S$ ,  $y$  is hashed by the  $k$  hash functions, and all locations  $y$  hashes to are checked. If any of the bits at the locations is 0,  $y$  is not in  $S$ , otherwise  $y$  is probably in  $S$ .

### 3.2. Secret Sharing

Secret sharing is a fundamental cryptographic primitive method of distributing secret among the participants each of whom is allocated a share of secret. In secret sharing scheme there is one dealer and  $n$  players. It allows a dealer to split a secret  $s$  into  $n$  shares such that the secret  $s$  can be recovered efficiently with any subset of  $t$  or more shares. With any subset of less than  $t$  shares, the secret is unrecoverable and the shares give no information about the secret. Such a system is called a  $(t, n)$ -secret sharing scheme.

When  $t = n$ , an efficient and secret sharing scheme can be obtained by simple (XOR) operations. The scheme works by generating  $n - 1$  random bit strings  $r_1, \dots, r_{n-1}$  of the same length as the secret  $s$ , and computing  $r_n = r_1 \oplus \dots \oplus r_{n-1} \oplus s$ . Each  $r_i$  is a share of the secret. It is easy to see that  $s$  can be recovered by computing  $r_1 \oplus \dots \oplus r_n$  and any subset of less than  $n$  shares reveals no information about the secret.

### 3.3. Oblivious Transfer

Oblivious transfer [16, 35] is a mechanism that allows a sender to send part of its input to a receiver in a manner that protects both parties. In oblivious transfer mechanism even though the sender sends part of its input, does not know which part the receiver receives and the receiver does not know any information about the remaining part of the sender's input. Generally, an oblivious transfer protocol can be denoted as  $OT_\ell^m$ . This notation means the sender holds  $m$  pairs  $\ell$ -bit strings  $(x_{j,0}, x_{j,1})$  for all  $j$  with  $0 \leq j \leq m - 1$ , while the receiver holds an  $m$ -bit selection string  $r = (r_0, \dots, r_{m-1})$ . At the end of the protocol execution, the receiver outputs  $x_{j,r_j}$  for  $0 \leq j \leq m - 1$ .

### 3.4. Garbled Bloom Filters

A garbled Bloom filter is a garbled version of the standard Bloom filter [11]. While analyzing simply, there is not any difference between a garbled Bloom filter and a Bloom filter but they have difference. In garbled Bloom filter, each location has a  $\lambda$ -bit string that is either a share of certain elements or a random string. It encodes a set of at most  $n$  elements in an array of length  $m$ , it also supports membership query with no false negative and negligible false positive. To add an element, the element is mapped by  $k$  independent uniform hash functions into  $k$  index numbers and the corresponding array locations are set. To query an element, the element is mapped by the same  $k$  hash functions into  $k$  index numbers and the corresponding array locations are checked. So, garbled Bloom filter seems to be exactly same with the standard Bloom filter but the difference between them is garbled Bloom filter stores share of data or random string in each slot whereas Bloom filter has only 1 or 0 in every slots.

## 4. Review of DCW13 Protocol

Dong, Chen and Wen [11] presented a new PSI protocol that is much more efficient than all the already existing PSI protocols. Their protocol is designed based on a novel two-party computation approach, which makes use of a new variant of Bloom filters which has been termed garbled Bloom filters, and they have referred the new approach as oblivious Bloom intersection. Their PSI protocol has been explained in the two versions: the basic one and the enhanced one. In the basic protocol they have proved the security in the semi-honest model and in the enhanced protocol the security has been proved in the malicious model. They have proved that the basic protocol has linear complexity (with a small constant factor) and relies mostly on symmetric key operations.

Not only efficiency there is another big advantage of this protocol, i.e., scalability. The computational, memory and communication complexities are all linear in the size of the input sets. They have presented that operations in the protocol can be performed in the SPMD (single program, multiple data) fashion that means little effort can also separate computation into a number of parallel tasks. So that their protocol can be executed with parallel processing capacity provided by current multi core CPUs and cloud computing environment. As multicore CPUs and cloud computing is supported by the protocol, it is can be implemented for Big Data oriented applications where data needs to be processed either in the parallel way or distributed way. Firstly, we will discuss the basic version of the

protocol.

## 4.1. Basic Protocol

The basic protocol has been discussed in the semi honest model and protocol is very simple. The client computes a Bloom filter which encodes its set  $C$  and the server computes a garbled Bloom filter which encodes its set  $S$ . To add an element  $x \in S$  to a Garbled Bloom Filter, first of all, split the element into  $k$   $\lambda$ -bit shares using the XOR-based secret sharing scheme. The element is also mapped into  $k$  index numbers and store single share in each location  $h_i(x)$ .

In a garbled Bloom filter, each location has a  $\lambda$ -bit string that is either a share of certain elements or a random string. For easy understanding, a share in a garbled Bloom filter is equivalent to a bit “1” in a Bloom filter, and a random string is equivalent to a bit “0”. Same as the Bloom filters, there is no false negative when using a GBF because all shares of an encoded element are guaranteed to be retrievable and the XOR-based secret sharing scheme always produces the original element when all shares are available. After the data preparation from the client and the server, they execute oblivious transfer protocol. At the end of the execution of protocol, the client obtains a garbled Bloom filter with the intersection of data and the server learns nothing. Then the client queries the intersection garbled Bloom filter and obtains the intersection value. To query an element  $y$ , collect all bit strings at  $h_i(y)$  and XOR them together. If the result is  $y$  then  $y$  is in  $S$ , otherwise  $y$  is not in  $S$ . The correctness is obvious: if  $y \in S$ , the XOR operation will recover  $y$  from its  $k$  shares which are retrievable from the garbled Bloom filter by their indexes. If  $y \notin S$ , then the probability of the XOR result is the same as  $y$  is negligible in  $\lambda$ .

The execution of the basic protocol has been explained and termed as oblivious bloom intersection [11].

### Oblivious Bloom Intersection

1. The server’s private input is  $S$ , the client’s private input is  $C$ . The auxiliary inputs include the security parameter  $\lambda$ , the maximum set size  $n$ , the optimal Bloom filter parameters  $m$ ,  $k$  and  $H = \{h_0, \dots, h_{k-1}\}$ . The parameter  $k$  is set to be the same as the security parameter  $\lambda$ .
2. The client generates an  $(m, n, k, H)$ -BF that encodes its private set  $C$ , the server generates an  $(m, n, k, H, \lambda)$ -GBF that encodes its private set  $S$ . The client uses its Bloom filter as the selection string and acts as the receiver in an  $OT_m^\lambda$  protocol. The server acts as the sender in the OT protocol to send  $m$  pair of  $\lambda$ -bit strings  $(x_i, 0, x_i, 1)$  where  $x_i, 0$  is a uniformly random

string and  $x_i, 1$  is  $GBF_S[i]$ . For  $0 \leq i \leq m-1$ , if  $BF_C[i]$  is 0, then the client receives a random string, if  $BF_C[i]$  is 1 it receives  $GBF_S[i]$ . The result is  $GBF_{C \cap S}^\pi$ .

3. The client computes the intersection by querying all elements in its set against  $GBF_{C \cap S}^\pi$ .

## 4.2. Protocol with Malicious Security

In the basic protocol, oblivious transfer is the interaction between the client and the server. While going through the protocol, it looks like we can obtain fully secure protocol by replacing the semi honest oblivious transfer protocol with secure against malicious party. But that is not enough to prove the security against the malicious client. Fully secure oblivious transfer protocol can prevent malicious behaviors like changing the input during protocol execution but it is not sufficiently strong to prevent the malicious client from full universe attack.

In a full universe attack, a malicious client encodes the full universe of all possible elements in its Bloom filter and uses it in the PSI protocol to learn the server’s entire set. A Bloom filter can easily represent the full universe by setting all the bits to 1. This is a feature of Bloom filters and causes a problem while constructing a simulator for the client in the malicious model. Namely, when the adversary uses the all-one Bloom filter, the simulator needs to enumerate all elements in the universe and send them to the trusted party in the ideal process. Without making any assumptions, the universe is potentially too large and a polynomial time algorithm may fail to enumerate all elements.

To prevent the full universe attack, authors have added a step to make sure that the client’s Bloom filter is not all-one. The server uses a symmetric key block cipher to encrypt strings in its garbled Bloom filter before transferring them to the client which forces the client to behave honestly by splitting the key into  $m$  shares using a  $(m/2, m)$ -secret sharing scheme. The client uses the bit array in its Bloom filter as the selection string to receive the intersection garbled Bloom filter and the shares of the key. If the bit in the selection string is 0, the client receives a share of the key, if the bit is 1, the client receives an encrypted string in  $GBF_S$ . The intuition is that if the client cheats by using an all-one Bloom filter, it will not be able to gather enough shares to recover the key, hence cannot decrypt the encrypted GBF. In the protocol  $m = 2kn$  to make sure that the client’s Bloom filter has at least  $m/2$  0 bits to receive enough shares to recover the key. Since the client has at most  $n$  elements and each element needs to be hashed  $k$  times, then the number of 1 bits in BFC will never exceed  $kn = m/2$ , consequently the number of 0 bits will always be at

least  $m/2$ .

The steps added for the security will not affect the client’s privacy, but might affect the correctness of the protocol when malicious server sends wrong key shares or uses a different key to encrypt its GBF. The client cannot detect that because the key is random and the strings in the GBF will look random. To prevent this malicious behavior, the client is required to send  $m$   $\lambda$ -bit random strings  $(r_0, \dots, r_{m-1})$  to the server before oblivious transfer. For each  $GBF_S[i]$ , the server encrypts  $r_i \parallel GBF_S[i]$  ( $\parallel$  means concatenation) and sends the ciphertext. After the transfer, the client can recover the key and decrypt the received ciphertexts. When the server is honest, the client can correctly decrypt using the key it recovered and  $r_i$  should present in the decrypted message. For each garbled Bloom filter string the client received, the probability of the server getting away with cheating is  $2^{-\lambda}$ .

### 4.3. Complexity

They have discussed the computational, communication and memory complexity in their paper. To build  $BF_C$  or  $GBF_S$ , each party needs  $k \cdot n$  hash operations. Then the server needs  $\lambda$  public key operations and the client need  $2\lambda$  public key operations for the Naor-Pinkas OT [31], and both parties need  $m = kn \log_2 e \approx 1.44kn$  hash operations for the oblivious transfer extension. The client needs to keep a copy of the Bloom filter and a copy of the intersection Garbled Bloom filter which in total need at most  $(\lambda + 1)m$  bits. This can be optimized to  $(\lambda/2 + 1)m$  bits because the client can throw away the string received when  $BF_C[i] = 0$  and leave  $GBF_{C \cap S}^\pi[i] = NULL$ . The server needs to store the garbled Bloom filter that is  $\lambda \cdot m$  bits. The main data sent in the protocol is a bit matrix required by the oblivious transfer extension and the strings sent by the server. In total  $2\lambda \cdot m$  bits. All other communication costs are much less significant and does not need to be considered.

## 5. Data Management

We discuss several data management methods that can be applicable on the GBF in this section. Since the DCW13 protocol improved the efficiency, it motivates us to study properties of garbled Bloom filter. Different mathematical properties can be applied on the garbled Bloom filter. We are discussing addition of data in the GBF, removal of data from GBF and merging the data from two GBF into a single one. For the better performance of algorithm, we have modified the server’s preparation of the GBF from existing protocol. We have incorporated counter array for each data slot which will be useful

to identify the number of shared states in that particular slot. This counter value has greater impact while removing the data from the GBF and merging two GBF into a single one as it will let us know the number of shares of data in each location.

### 5.1. GBF Preparation

For the preparation of a GBF as explained in Algorithm 1 below, we first create an empty GBF and initialize each location values to NULL. To add  $x \in S$ , we will split  $x$  into  $k$  shares and store the shares in  $GBF_S[h_i(x)]$ . We have counter  $C$  which will be initiated by 0 and will be incremented for each shares on the particular location. We also know in this process, some location  $j = h_i(x)$  may have been occupied by a previously added element already. For this scenario, we have to reuse the existing share stored at  $GBF_S[j]$ . After adding all elements in  $S$ , we generate and store random  $\lambda$ -bit strings at all NULL locations.

---

#### Algorithm 1 Build GBF with Counter

---

**Input:** A set  $S, n, m, k, \lambda, C, H = \{h_0, \dots, h_{k-1}\}$   
**Output:** An  $(m, n, k, H, \lambda, C)$ -Garbled Bloom filter  $GBF_S$

```

1: for  $i = 0$  to  $m - 1$  do
2:    $GBF_S[i] = NULL$ ;
3: end for
4: for all  $x \in S$  do
5:    $emptySlot = -1, finalShare = x$ 
6:   for  $i = 0$  to  $k - 1$  do
7:      $j = h_i(x)$ 
8:      $C[i] = 0$ 
9:     if  $GBF_S[j] == NULL$  then
10:      if  $emptySlot == -1$  then
11:         $emptySlot = j$ 
12:      else
13:         $GBF_S[j] \leftarrow \{0, 1\}^\lambda$ ;
14:         $finalShare = finalShare \oplus GBF_S[j]$ 
15:         $C[i] ++$ 
16:      end if
17:    else
18:       $finalShare = finalShare \oplus GBF_S[j]$ 
19:       $C[i] ++$ 
20:    end if
21:  end for
22:   $GBF_S[emptySlot] = finalShare$ 
23:  for  $i = 0$  to  $m - 1$  do
24:    if  $GBF_S[i] == NULL$  then
25:       $GBF_S[i] \leftarrow \{0, 1\}^\lambda$ 
26:       $C[i] = 0$ 
27:    end if
28:  end for
29: end for

```

---

## 5.2. Data Addition on GBF

Data addition algorithm 2 below helps the server to add the data over the existing GBF. Basically, data addition on the GBF process is somewhat similar to the preparation of new GBF but the only difference is, to add data over the already existing data filled locations in the GBF. We have  $GBF_S$  with data or random value filled in each locations as input. We will obtain exact same size of  $GBF_S$  on output but with more data added on it. To add  $x \in S$ , first split  $x$  into  $k$  shares and store the shares in  $GBF_S[j]$ . We have counter  $C$  value from the input  $GBF_S$  which will be incremented for each shares on the particular location. We also know location  $j = h_i(x)$  has value and we have to reuse the existing share stored at  $GBF_S[j]$ . After adding all elements in  $S$  to the new  $GBF_S$ , if there are any NULL locations, we generate and store random  $\lambda$ -bit strings to ensure that none of the location in the GBF are blank for the purpose of security.

---

### Algorithm 2 Add Data on GBF

---

**Input:** An  $(m, n, k, H, \lambda, C)$ -garbled Bloom filter  $GBF_S$

**Output:** An  $(m, n, k, H, \lambda, C)$ -garbled Bloom filter  $GBF_S$

```

1: for all  $x \in S$  do
2:    $emptySlot = -1, finalShare = x$ 
3:   for  $i = 0$  to  $k - 1$  do
4:      $j = h_i(x)$ 
5:      $C[i] = 0$ 
6:     if  $GBF_S[j] == NULL$  then
7:       if  $emptySlot == -1$  then
8:          $emptySlot = j$ 
9:       else
10:         $GBF_S[j] \leftarrow \{0, 1\}^\lambda$ 
11:         $finalShare = finalShare \oplus GBF_S[j]$ 
12:         $C[i] ++$ 
13:      end if
14:    else
15:       $finalShare = finalShare \oplus GBF_S[j]$ 
16:       $C[i] ++$ 
17:    end if
18:  end for
19:   $GBF_S[emptySlot] = finalShare$ 
20:  for  $i = 0$  to  $m - 1$  do
21:    if  $GBF_S[i] == NULL$  then
22:       $GBF_S[i] \leftarrow \{0, 1\}^\lambda$ 
23:       $C[i] = 0$ 
24:    end if
25:  end for
26: end for

```

---

## 5.3. Data Deletion from GBF

To delete an element from a garbled Bloom filter, we can find one index in the garbled Bloom filter that the element hashes to and no other elements

do. This index is allocated exclusively to the element which has to be deleted. Then we can replace the string at this index with a random string. The idea is that we test set membership by XORing all shares, this idea delete one share of the element from the garbled Bloom filter, so set membership query will always return false. It does not affect other elements since this share is not used by other elements.

---

### Algorithm 3 Remove Data from GBF

---

**Input:** An  $(m, n, k, H, \lambda, C)$ -garbled Bloom filter  $GBF_S$

**Output:** An  $(m, n, k, H, \lambda, C)$ -garbled Bloom filter  $GBF_S$

```

1: for all  $x \in S$  do
2:   for  $i = 0$  to  $k - 1$  do
3:      $share = x$ 
4:      $j = h_i(x)$ 
5:     if  $C[j] == 1$  then
6:        $share \leftarrow \{0, 1\}^\lambda$ 
7:        $GBF_S[j] = share$ 
8:        $C[j] = 0$ 
9:     end if
10:  end for
11: end for

```

---

## 5.4. GBF Merging

To merge two different GBFs into a single one. We can simply get the union of two different GBFs. We have two input GBFs and single output GBF. For each data we check whether every slot has data or random value. If for any particular slot, both GBF have data, we do XOR of them and allocate it to the same index value in output GBF. If any of the slot from input GBF have random value we check the corresponding slot value of the another input GBF, if it has data, it will be restored in the output GBF else random value will be generated and stored.

## 5.5. Memory and Communication Complexity

Even we have implemented data management techniques with slight modifications over DCW13 Protocol, number of steps of execution doesn't vary much with their approach so that time and space complexity value remains same with DCW13 protocol. So, we can write memory complexity as  $\lambda \cdot m$  bits and communication complexity as  $2\lambda \cdot m$  bits .

## 6. Performance Evaluation

In this section we discuss the performance estimation of results of our updated technique of GBF

---

**Algorithm 4** Merge GBF

---

**Input:** Two  $(m, n, k, H, \lambda, C)$   $GBF_{S_1}, GBF_{S_2}$ **Output:** An  $(m, n, k, H, \lambda, C)$ -garbled Bloom filter  $GBF_S$ 

```
1: for  $i = 0$  to  $k - 1$  do
2:   if  $C1[i] \neq 0 \ \&\& \ C2[i] \neq 0$  then
3:      $GBF_S[i] \leftarrow GBF_{S_1}[i] \oplus GBF_{S_2}[i]$ 
4:      $C[i] \leftarrow C1[i] + C2[i]$ 
5:   else
6:     if  $C1[i] == 0 \ \&\& \ C2[i] \neq 0$  then
7:        $GBF_S[i] \leftarrow GBF_{S_2}[i]$ 
8:        $C[i] \leftarrow C2[i]$ 
9:     else
10:      if  $C1[i] \neq 0 \ \&\& \ C2[i] == 0$  then
11:         $GBF_S[i] \leftarrow GBF_{S_1}[i]$ 
12:         $C[i] \leftarrow C1[i]$ ;
13:      else
14:         $GBF_S[i] \leftarrow \{0, 1\}^\lambda$ 
15:         $C[i] = 0$ 
16:      end if
17:    end if
18:  end if
19: end for
```

---

preparation and modification. We have not executed the PSI protocol but executed the source code to create Garbled Bloom Filter from Dong et al's protocol and our updated version. We have performed the experiment in Macbook Pro laptop with an Intel core  $i51.4 \text{ GHz}$  CPU, 4 GB RAM and runs Mac OS X 10.7. We have chosen the GBF parameters  $k = \lambda$  and set  $m$  to be the optimal value  $k_n \log_2 e$ . So for, at 80-bit security  $k = \lambda = 80$ , and when  $n = 220$ ,  $m = 120795960$ . While creating GBF using the Dong et. al's protocol with the server data set size  $S = 10000$ , total execution time was 5.88 sec and with the execution of algorithm 1 explained above it took 7.02 sec. So, even implementing counter while preparing GBF doesn't alter the performance of PSI protocol. As our paper is focused on the data management techniques over the Dong et al's protocol, security analysis will be same with their protocol. For PSI protocol execution using our modified Garbled Bloom Filter, it might be slightly alter the performance of the protocol.

## 7. Conclusion and Future Work

In this paper we have improved over efficient and scalable PSI protocol based on oblivious Bloom intersection by Dong et al. Their protocol depends mostly on efficient symmetric key operations and the operations can be parallelized easily. They presented two variants of the protocol: the basic one is secure in the semi-honest model and the enhanced one is secure in the malicious model. Comparing with other protocols performance evaluation and comparison results show that their protocol is

magnitude faster than previous protocols. The efficiency make it suitable for large scale privacy preserving data processing. Based on their protocol, we have presented data management methods in Garbled Bloom Filter, which will enable us to perform the addition, the deletion of data among the GBF and merging two GBFs into a single one. We believe these data management methods presented will enhance more in the execution of PSI.

## References

- [1] B. Applebaum, M. Caesar, M. J. Freedman, J. Rexford and H. Ringberg. Collaborative, privacy-preserving data aggregation at scale. In *Proc. 10th Intern'l Symp. Privacy Enhancing Technologies*, LNCS 6205, pp.56–74, 2010.
- [2] G. Ateniese, E. De Cristofaro, and G. Tsudik. (If) size matters: Size-hiding private set intersection. In *Proc. 14th Intern'l Conf. Practice and Theory in Public Key Cryptography*, LNCS 6571, pp.156–173, 2011.
- [3] P. Baldi, R. Baronio, E. De Cristofaro, P. Gasti, and G. Tsudik. Countering gattaca: efficient and secure testing of fully sequenced human genomes. In *Proc. 18th ACM Conf. Computer and Communications Security*, pp.691–702, 2011.
- [4] D. Beaver. Correlated pseudo randomness and the complexity of private computations. In *Proc. 28th Annual ACM Symp. Theory of Computing*, pp.479–488, 1996.
- [5] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proc. 1st ACM Conf. Computer and Communications Security*, pp.62–73, 1993.
- [6] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
- [7] P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. H. M. Smid, and Y. Tang. On the false-positive rate of bloom filters. *Information Processing Letters*, 108(4):210–213, 2008.
- [8] E. Bursztein, J. Lagarenne, M. Hamburg, and D. Boneh. OpenConflict: Preventing Real Time Map Hacks in Online Games. In *Proc. 32nd IEEE Symp. Security and Privacy*, pp.506–520, 2011.
- [9] J. Camenisch and G. M. Zaverucha. Private intersection of certified sets. In *Proc. 13th Intern'l Conf. Financial Cryptography and Data Security*, LNCS 5628, Springer, pp.108–127, 2009.
- [10] R. Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.
- [11] C. Dong, L. Chen, and Z. Wen, When private set intersection meets big data : An efficient and scalable protocol. In *Proc. 2013 ACM SIGSAC Conf. Computer and Communications Security*, pp.789–800, 2013.
- [12] E. De Cristofaro, J. Kim, and G. Tsudik. Linear-complexity private set intersection protocols secure

- in malicious model. In *Proc. ASIACRYPT 2010*, LNCS 6477, pp.213–231, 2010.
- [13] E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In *Proc. 14th Intern'l Conf. Financial Cryptography and Data Security*, LNCS 6052, pp.143–159, 2010.
- [14] E. De Cristofaro and G. Tsudik. Experimenting with fast private set intersection. In *Proc. 5th Intern'l Conf. Trust and Trustworthy Computing*, LNCS 7344, pp.55–73, 2012.
- [15] D. Dachman-Soled, T. Malkin, M. Raykova, and M. Yung. Efficient robust private set intersection. In *Proc. 7th Intern'l Conference on Applied Cryptography and Network Security*, LNCS 5536, pp.125–142, 2009.
- [16] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.
- [17] M. Fischlin, A. Lehmann, T. Ristenpart, T. Shrimpton, M. Stam, and S. Tessaro. Random oracles with(out) programmability. In *Proc. ASIACRYPT 2010*, LNCS 6477, pp.303–320, 2010.
- [18] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Proc. EUROCRYPT 2004*, LNCS 3027, pp.1–19, 2004.
- [19] O. Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge Univ. Press, 2004.
- [20] C. Hazay and Y. Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *Proc. 5th Conf. Theory of Cryptography*, LNCS 4948, pp.155–175, 2008.
- [21] C. Hazay and K. Nissim. Efficient set operations in the presence of malicious adversaries. In *Proc. 13th Intern'l Conf. Practice and Theory in Public Key Cryptography*, LNCS 6056, pp.312–331, 2010.
- [22] Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols. In *19th Network and Distributed Security Symp.*
- [23] S. Jarecki and X. Liu. Efficient oblivious pseudo-random function with applications to adaptive and secure computation of set intersection. In *Proc. 6th Theory of Cryptography Conf.*, LNCS 5444, pp.577–594, 2009.
- [24] S. Jarecki and X. Liu. Fast secure computation of set intersection. In *Proc. 7th Intern'l Conf. Security and Cryptography for Networks*, LNCS 6280, pp.418–435, 2010.
- [25] F. Kerschbaum. Outsourced private set intersection using homomorphic encryption. In *Proc. 7th ACM Symp. Information, Computer and Communications Security*, pp.85–86, 2012.
- [26] L. Kissner and D. X. Song. Privacy-preserving set operations. In *Proc. CRYPTO 2005*, LNCS 3621, pp.241–257, 2005.
- [27] M. Li, N. Cao, S. Yu, and W. Lou. FindU : Privacy-preserving personal profile matching in mobile social networks. In *Proc. IEEE INFOCOM 2011*, pp.2435–2443, 2011.
- [28] D. Many, M. Burkhart, and X. Dimitropoulos. Fast private set operations with sepi. TIK Report 345, ETH Zurich, Mar 2012.
- [29] G. Mezzour, A. Perrig, V. D. Gligor, and P. Papadimitratos. Privacy-preserving relationship path discovery in social networks. In *Proc. 8th Intern'l Conf. Cryptology and Network Security*, LNCS 5888, pp.189–208, 2009.
- [30] S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar, and N. Borisov. Botgrep: Finding P2P bots with structured graph analysis. In *USENIX Security Symp. 2010*, pp.95–110, 2010.
- [31] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *Proc. 12th Annual ACM-SIAM Symp. Discrete Algorithms*, pp.448–457, 2001.
- [32] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh. Location privacy via private proximity testing. In *18th Network and Distributed System Security Symp. (NDSS 2011)*.
- [33] B. Pinkas, T. Schneider, and M. Zohner. Faster Private Set Intersection based on OT Extension. In *USENIX Security Symp. 2014*.
- [34] O. Papapetrou, W. Siberski, and W. Nejdl. Cardinality estimation and dynamic length adaptation for bloom filters. *Distributed and Parallel Databases*, 28(2–3):119–156, 2010.
- [35] M. O. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.
- [36] A. Shamir. Identity-based crypto systems and signature schemes. In *Proc. CRYPTO '84*, LNCS 196, pp.47–53, 1985.